Какво прави GPU-то по време на една игра

Ангел Дончев Ангелов

Агенда

- **Интро**:
  - Кой съм аз?
  - Какво правя?
  - Защо представям тази тема?

- **Графика**
  - Какво е графичен фрейм?
  - Различни методи на създаване
  - Защо в/у GPU-то?

- **Практична Графика**
  - „Картон“ за рисуване
  - Триъгълник
  - Триъгълници
  - Камера
  - Текстури

Кой съм аз?

- Здравейте! Аз съм Ангел Дончев Ангелов
- 3-та година програмист в „Бреда Университет за Приложни Науки“
- Програмата се нарича Creative Media and Game Technologies
  - Програмисти 👨‍👩‍💻
  - Артисти 👨‍🎨 🧑‍💻
  - Дизайнери 🕵️ 👨‍💻

🎮🎮 Учим се да правим игри! 🎮🎮

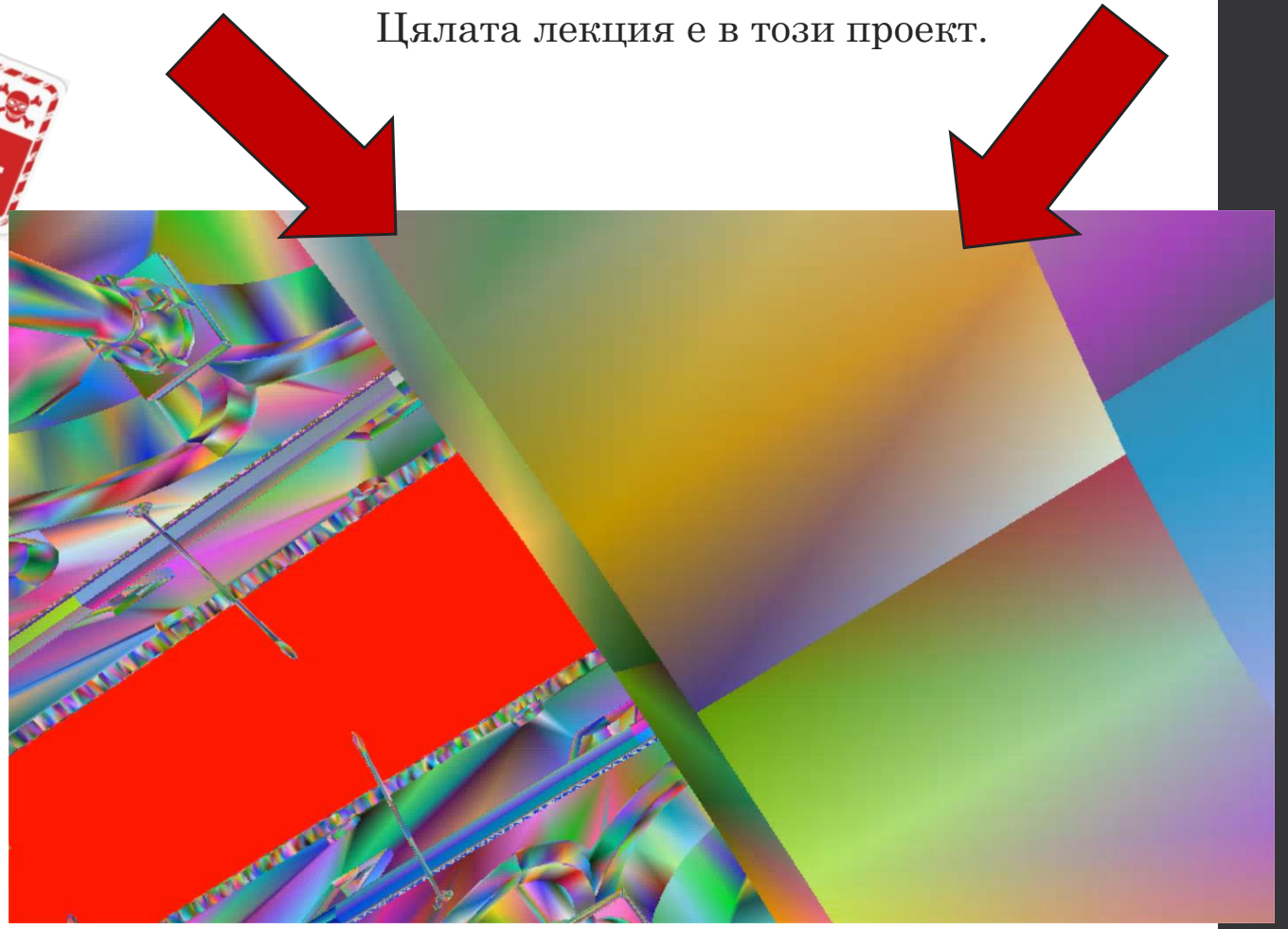
Breda University of Applied Sciences

Какво точно правя?

- *(Най – вече)* Графично програмиране!
- Няколко интересни проекти: **2 CPU Renderers**

CPU Ray Tracer

Цялата лекция е в този проект.



CPU Rasterizer

Какво точно правя?

- *(Най – вече)* Графично програмиране!
- Няколко интересни проекти: **2 GPU Renderers (for Game Engines)**

~~"Carrr" Engine~~

(NDA)

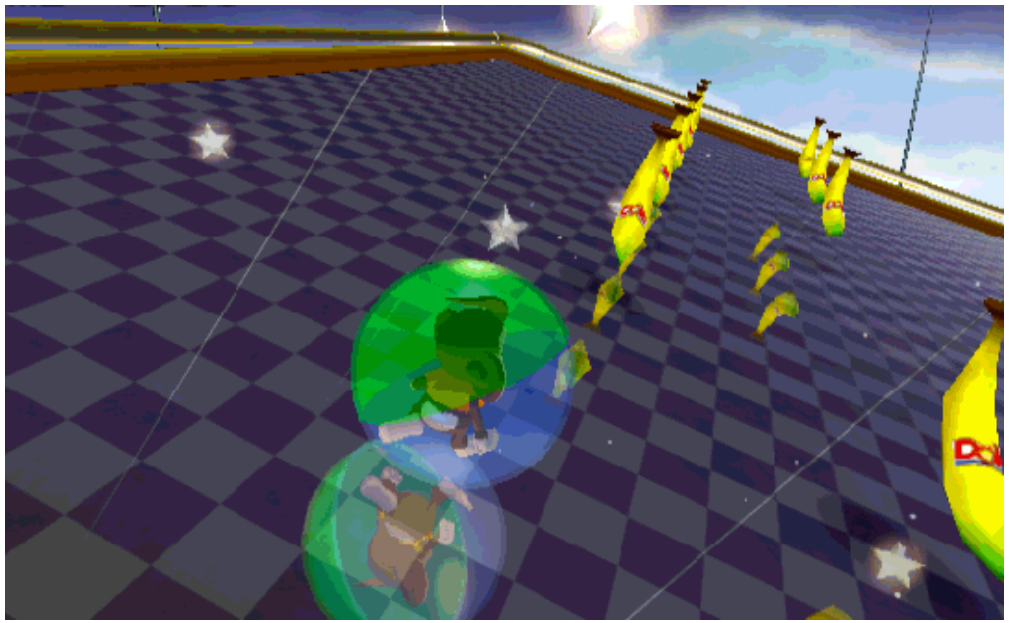
HELLBOUND

(NDA, но може да я играете от Itch.io)

Какво точно правя сега?

- Едногодишен проект:
- Отбор от 9 програмисти:
  - 4 Графика
  - 2 Engine
  - 2 Физика
  - 1 Инструменти

- Windows и PlayStation 5



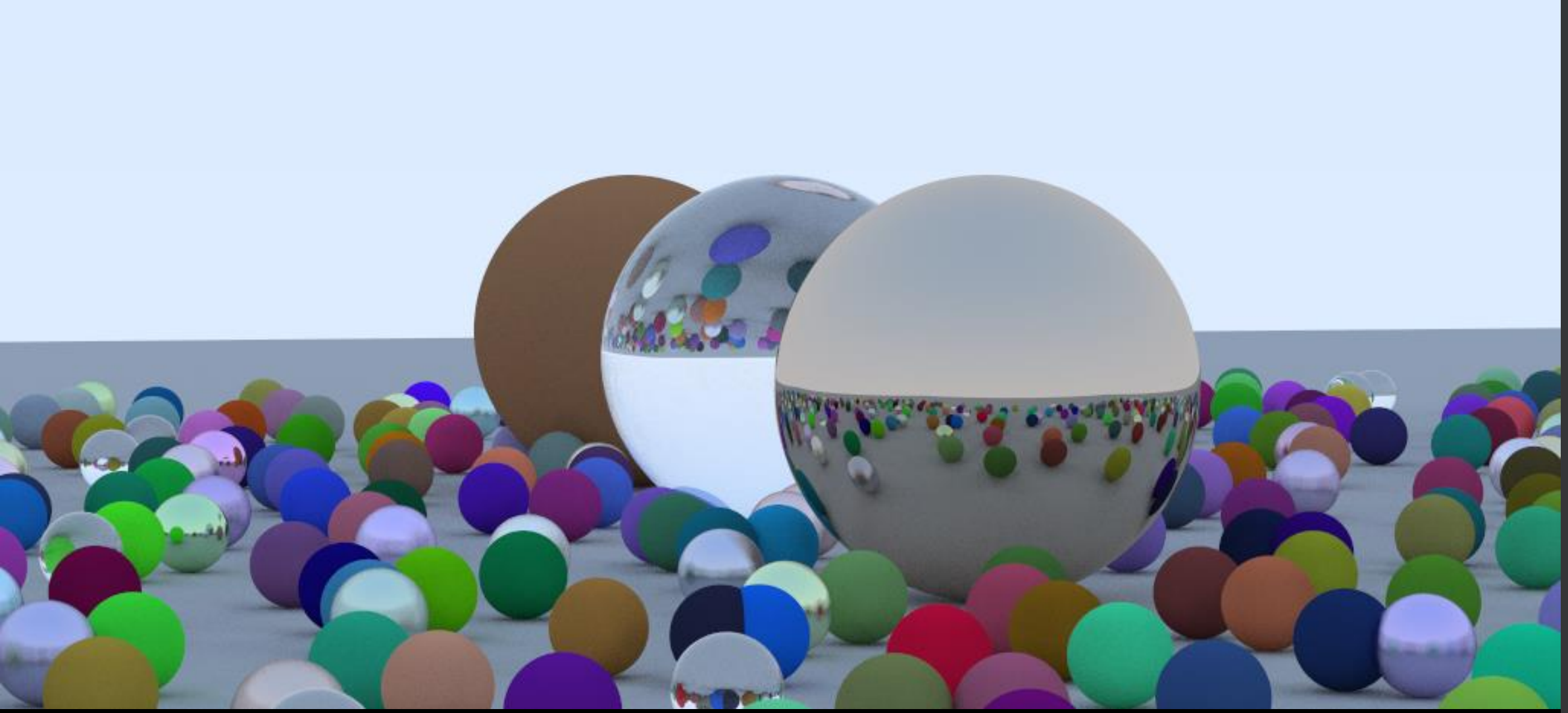
Една академична година по - късно

Super Monkey Ball:
Ray Traced
and
Re-Imagined

COMING SOON!

Защо представям тази тема?

- “Как става всичко това?”

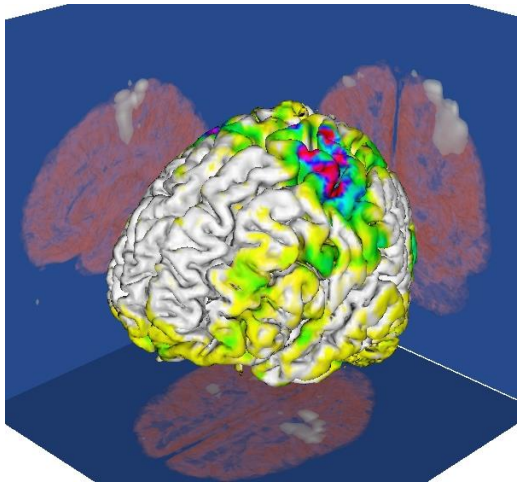


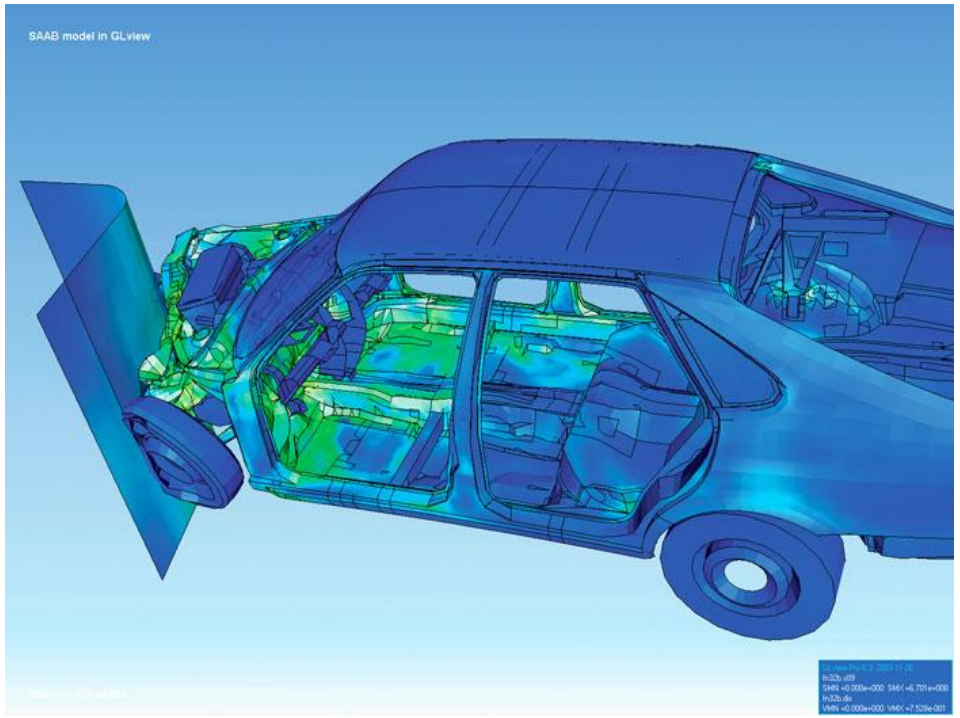
Графика - Интро

Снимка:
„Ray Tracing in One Weekend“
– Peter Shirley

Какво е компютърна графика?

- Много широка сфера
- Някакъв вид визуализация, създадена от компютър
- За нас –
  - 3Д сцена
  - Имитира истинския свят





Какво е компютърна графика?

Интерактивност

- Кой е ресурса, който го имаме при изграждане филми, но го нямаме при правенето на игри?

- Време

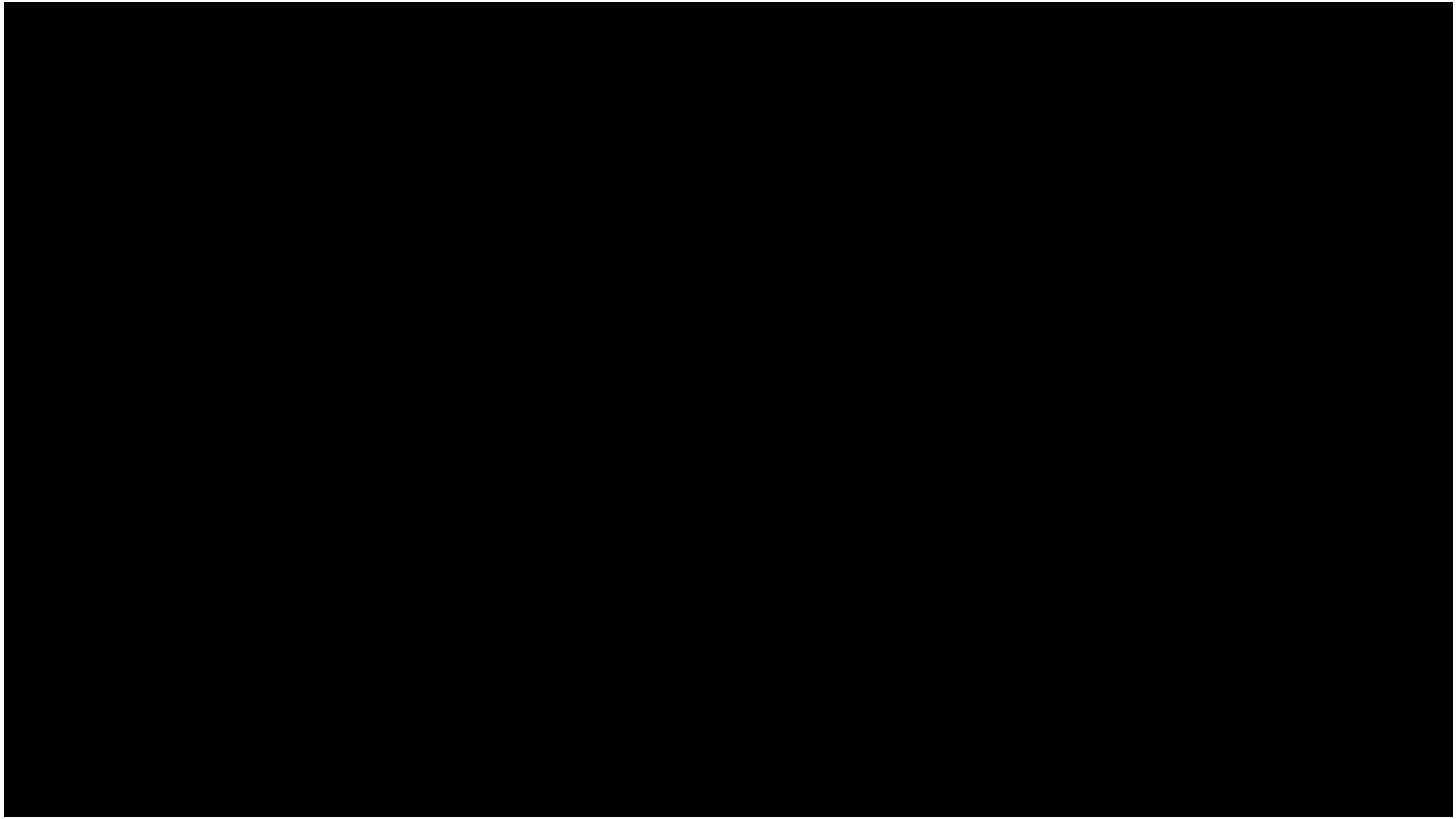
- Създаването на филма „Лука“:
  - 1 кадър = 50 часа рендъринг работа
  - 24 кадри в секунда
  - 95 минутен филм

780 години работа!

Нашите картини трябва да са **интерактивни**: 60 кадъра в секунда

Интерактивност

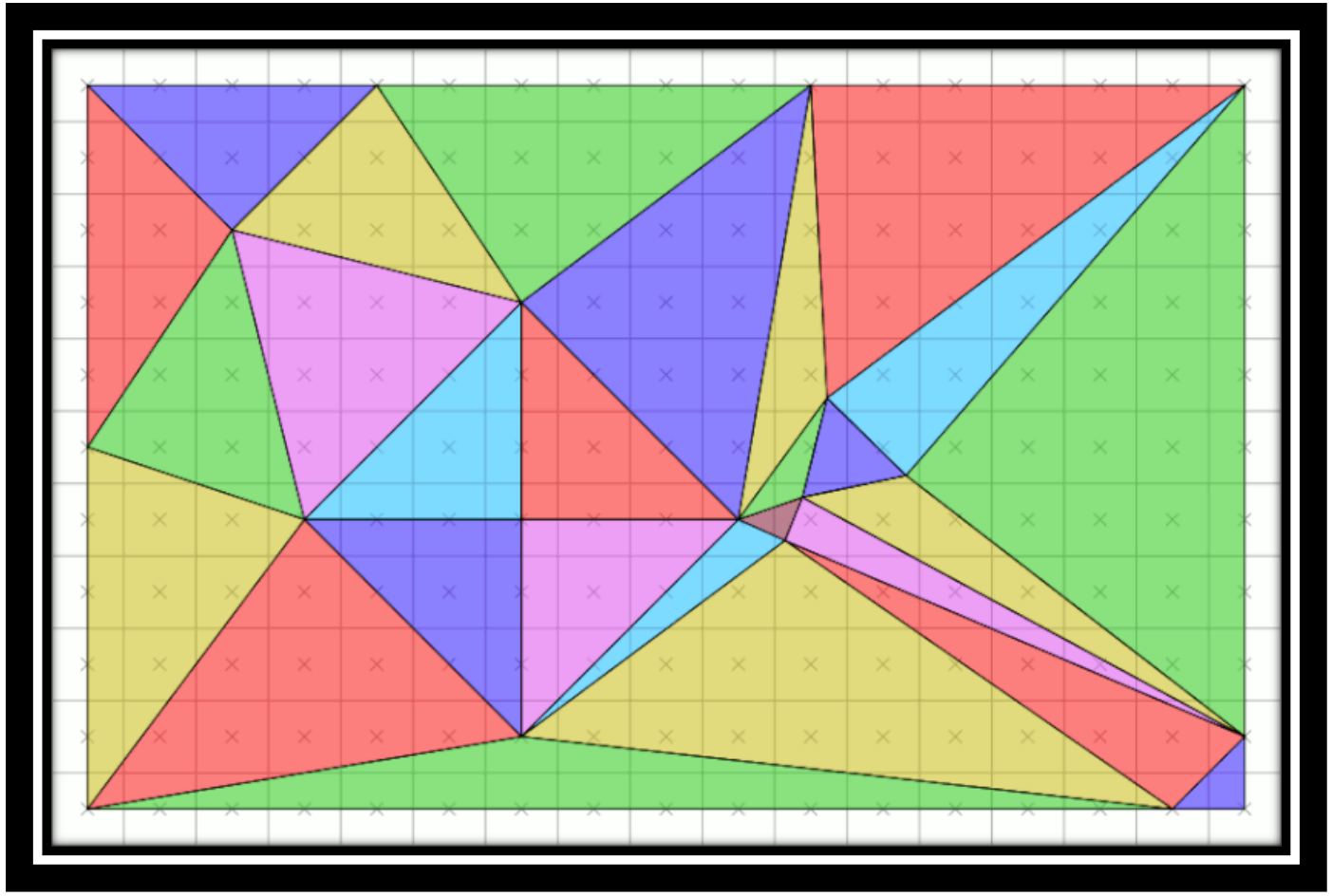
Линк към оригиналното видео



Стандартни методи: Растеризация

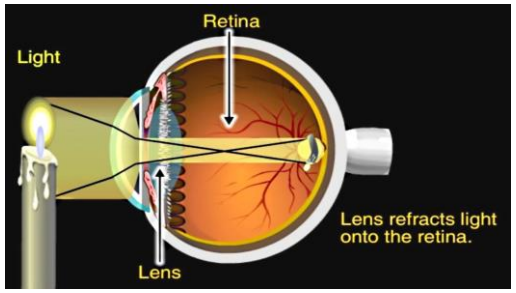
- Растеризация е процесът на преобразуване на векторни изображения в растерни изображения, които се състоят от множество пиксели.

- Пример:
  - 17 X 11 екран
  - Проверяваме дали средата на триъгълника е в триъгълника. Ако е, този пиксел е с цвета на триъгълника.



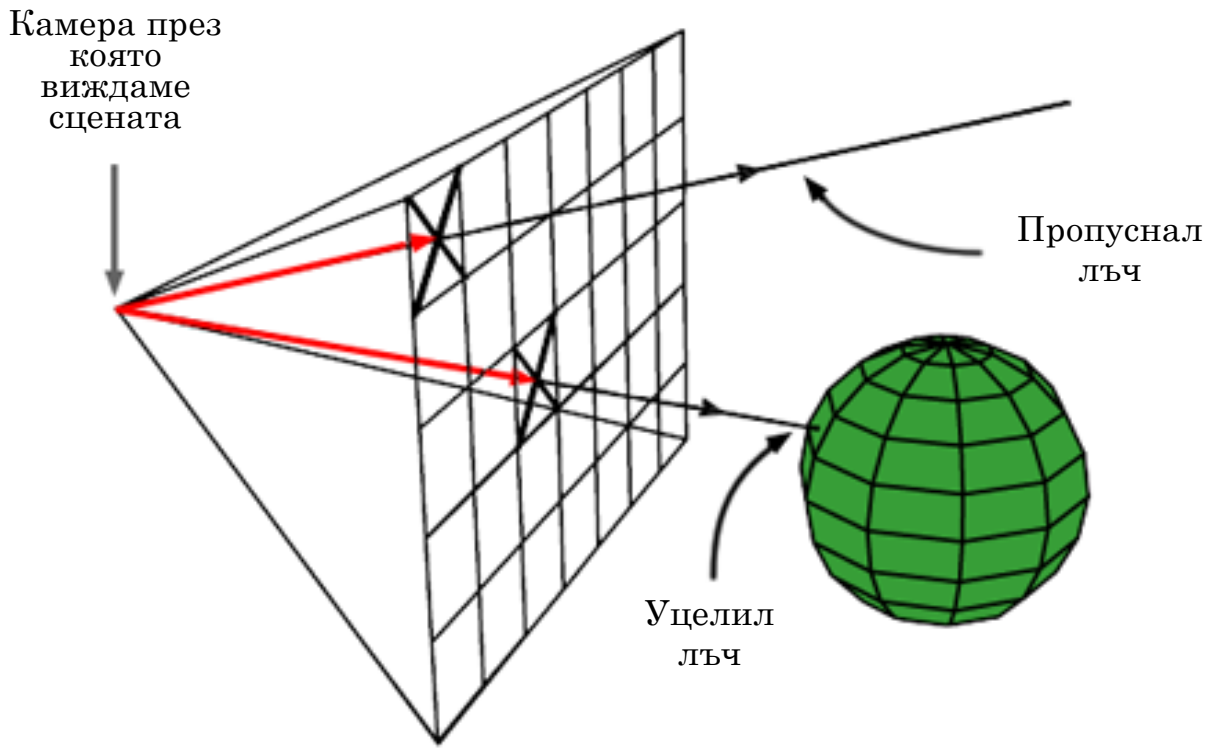
Стандартни методи: Ray Tracing

- Ray Tracing-а е метод при който изтрелваме лъч от камерата към квадрата, където е проектиран екрана ни.



  - Проверяваме дали лъча е уцелил някакъв обект в екрана.
  - Ако го е уцелил, пикселът е цвета на материала.
  - Ако лъча пропусне му задаваме цвета на фона.

- Симулира истинския живот.



Стандартни методи: Невронна мрежа

- Много нов похват
- Използва невронна мрежа, за да запамети сцената.
- Линк за тези, които им е интересно

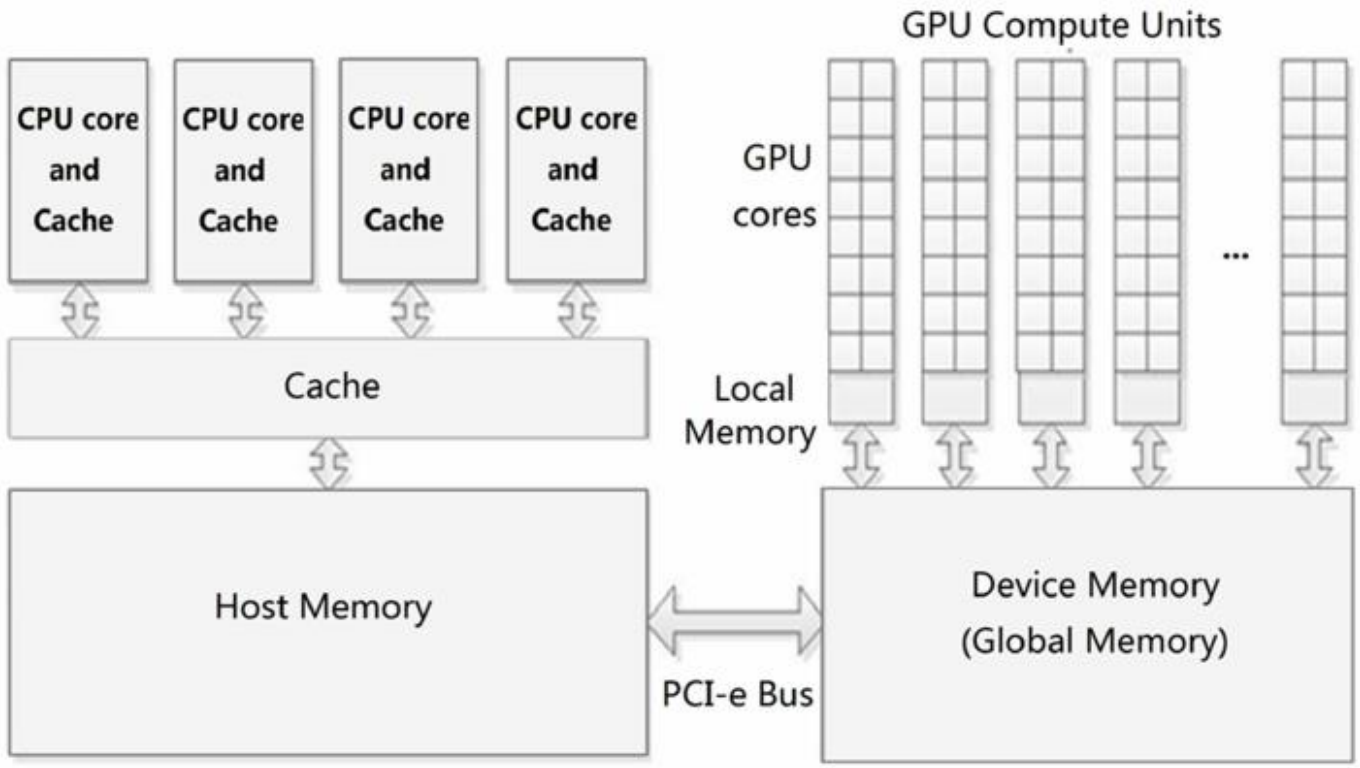
Защо на GPU-то?

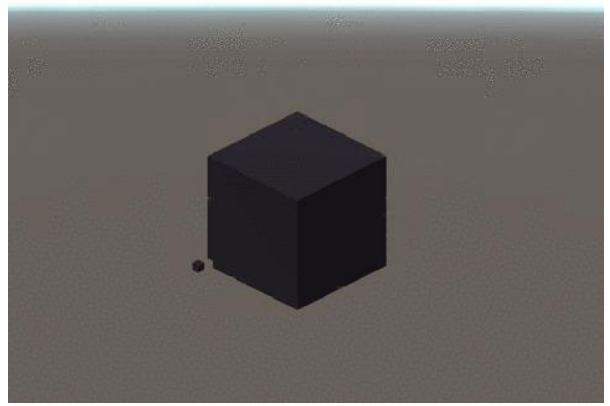
- Защото е бързо!
  - Хиляди ядра – CPU 2 – 64 ядра
  - Паралелно решаване на задачи
  - Бърза Floating Point математика

  - FLOPS - Floating point operations per second
    - CPU Intel® Xeon® D-1749NT – 416 G-FLOPS
    - GPU Nvidia RTX 3090 – 35,000 G-FLOPS
    - 1 Gigaflop = 1,000,000,000 Flop

  - Host and Device термините
    - Host е CPU-то – Мениджър
    - Device е GPU-то – Техничарч

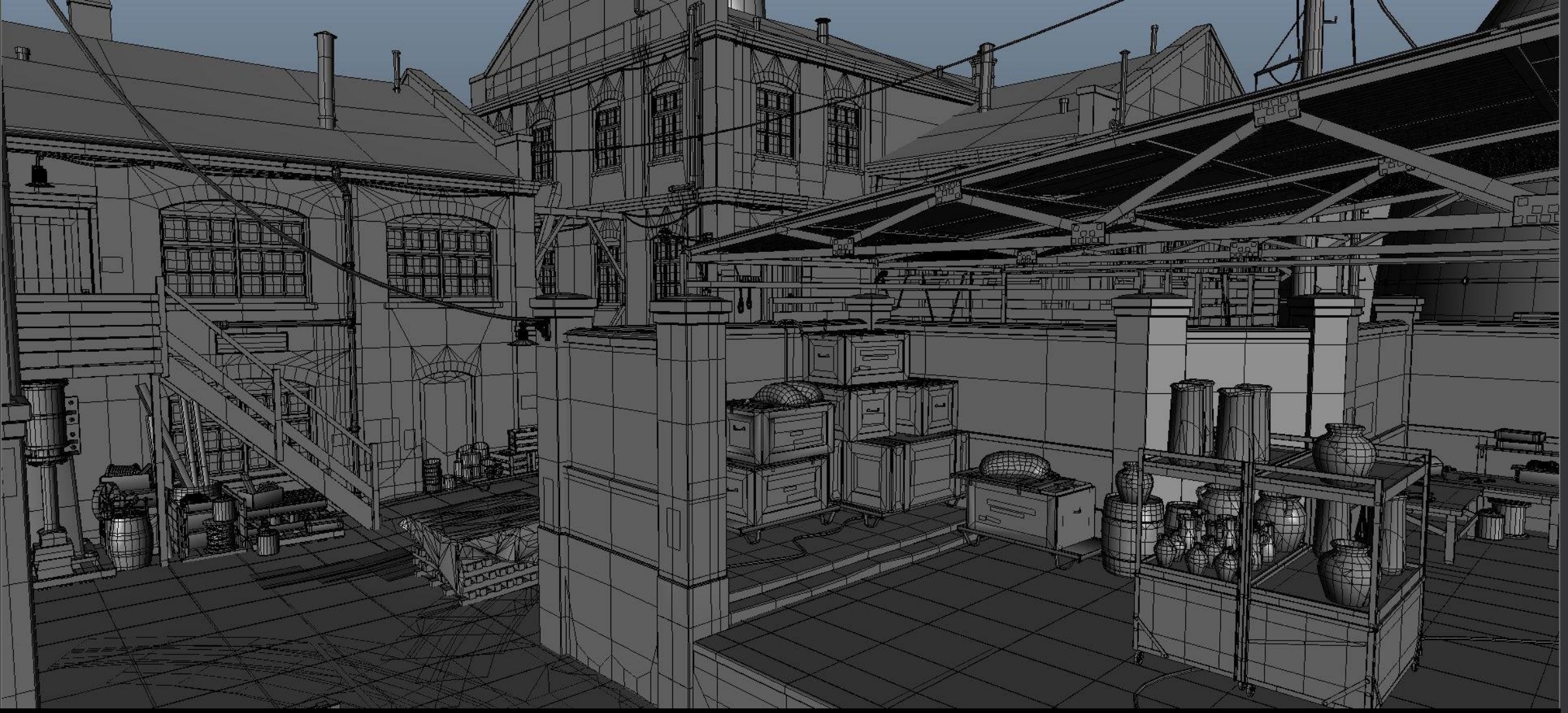
- Освобождава CPU-то да се занимава с по – важна работа:
  - Физика
  - Изкуствен интелект
  - Менежиране на ресурси





Как програмираме в/у GPU-то?

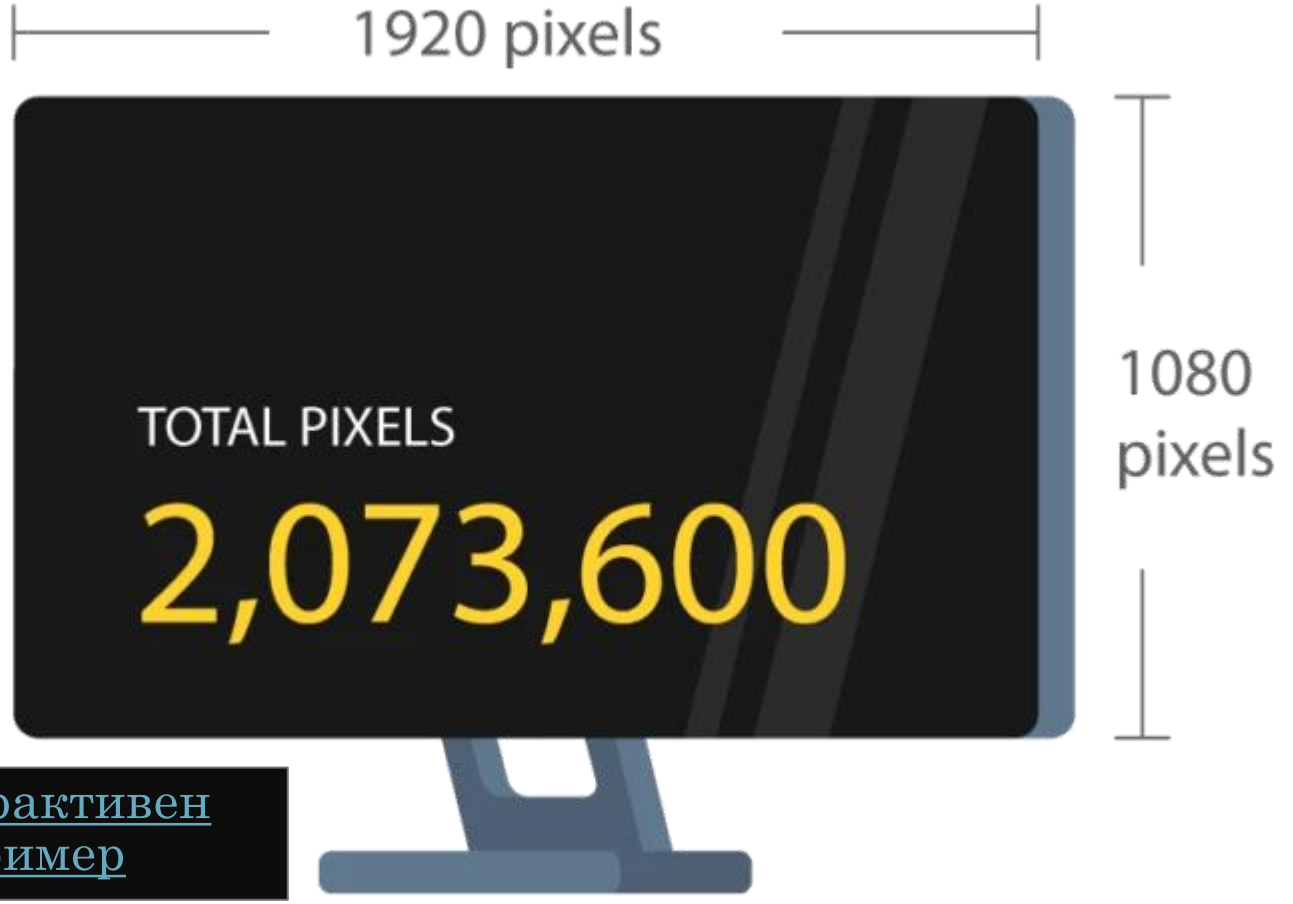
- С Графични API-та
  - Какво са те?
- OpenGL
  - Най – лесният low-level Графичен API
  - Перфектен за начинаеши
- DirectX 12
  - Много сложен / ниско ниво API
  - Само за Microsoft машини (включва Xbox)
  - Много развит
- Vulkan
  - Много сложен / ниско ниво API
  - Дава най – много контрол, защото не приема неща за даденост.
  - Супер между-платформен
- Metal
  - Перфектен за Екосистемата на Apple
  - Единствено за Apple устройства
- Play Station 5
  - Под NDA, не е публичен
  - Специфично направен за PlayStation 5-цата
  - Много специализирани инструменти за всичко



Графика – В практиката

Да започнем от нулата

- Преди да започнем да рисуваме ни трябва някакъв картон.
- В контекста на игри, често е с размера на целия екрана в отделен прозорец
- Има много имена: Текстура, Swap-chain, Frame buffer, etc.
- Последното място, където показваме изображението



Какво е пиксел?

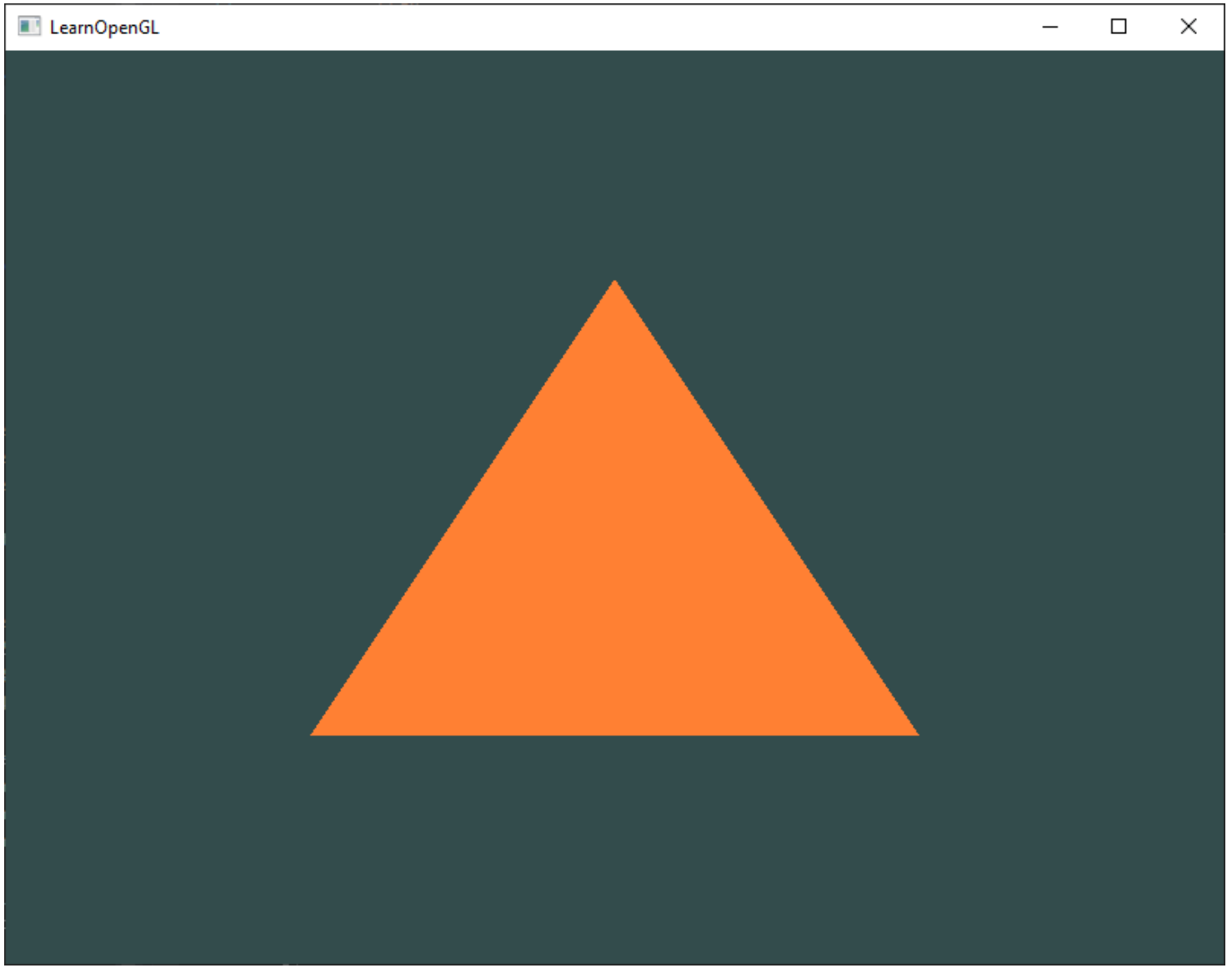
- Точка на екрана - цвят може да контролираме,
- В програмирането един пиксел на екрана най – често се дефинира като цяло число от 4 байта.
- 1 байт е 8 бита, които се използват, за да се запази число от 0 – 255
- 4 канала - Всеки канал е 1 байт,. Червено, Зелено, Синьо и Алфа

Интерактивен пример

„Hello Triangle”

- “Hello world” на Графичното програмиране
- 3 точки / върtекси

```
vec2(0.5, 0.75),
vec2(0.3, 0.25),
vec2(0.7, 0.25)
```



В зависимост от Графичното API, което използваме, това може да отнеме 2000 линии код.

DirectX® 12

Vulkan®

Триъгълник към Пиксел

- Edge функцията – Pineda, J. (1988). *A parallel algorithm for polygon rasterization.*

$$XY \times XP = (v_1 - v_0) \times (p - v_0)$$

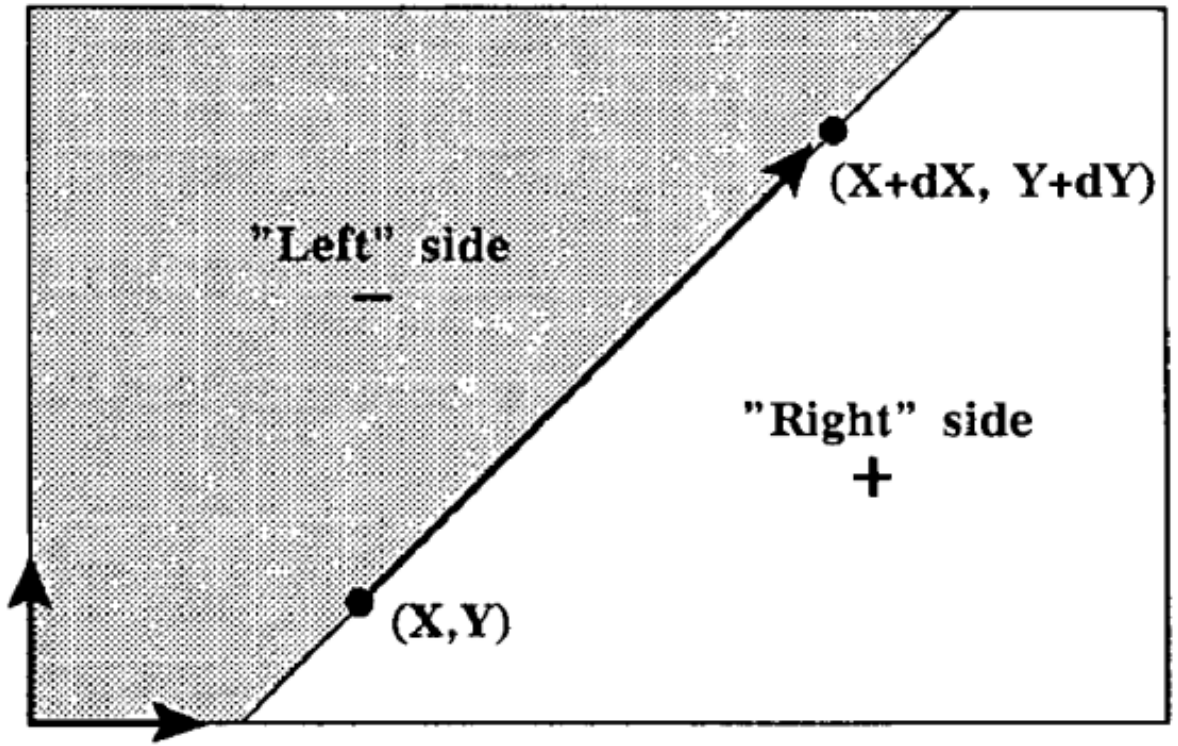
$$(v_1 - v_0).x * (p - v_0).y - (v_1 - v_0).y * (p - v_0).x$$

- Векторното произведение на точките XY и XP.

- Desmos Пример

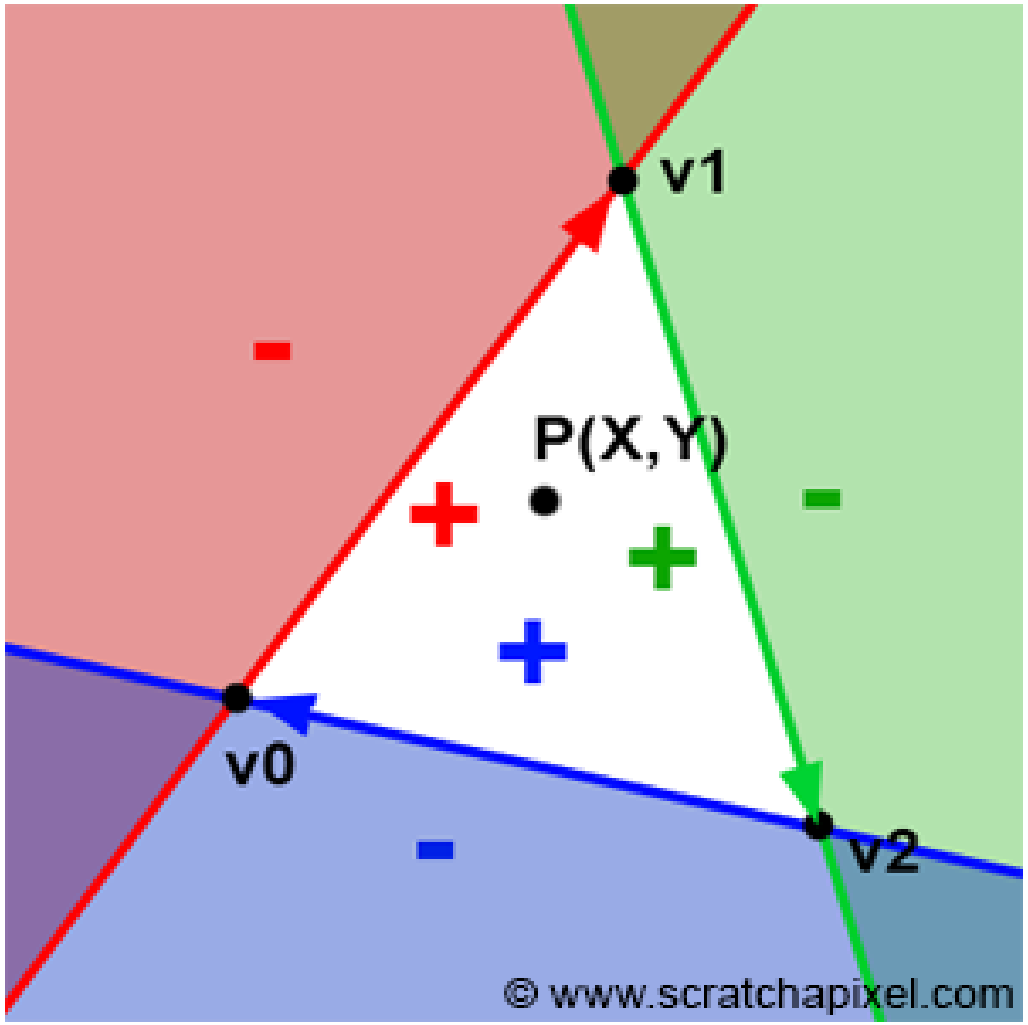
- C++ Пример:

```
 9
10 float edge_func(Vec2 p, Vec2 v0, Vec2 v1) {
11     Vec2 v0_p = {p.x - v0.x, p.y - v0.y};
12     Vec2 v0_v1 = {v1.x - v0.x, v1.y - v0.y};
13
14     return v0_p.x * v0_v1.y - v0_p.y * v0_v1.x;
15 }
16
```



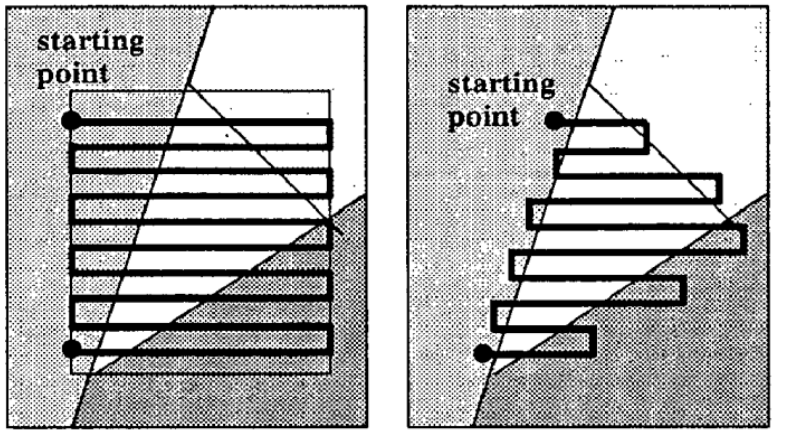
Триъгълник към Пиксел

- Как прилагаме това към триъгълник?
  Правим го 3 пъти!



- **Внимание!** При Векторното произведение реда има значение, ако го обърркаме при която и да е от линиите няма да ни излезе триъгълника.

- Малка оптимизация:



Traversing the Bounding Box *A More Efficient Traversal Algorithm*

Figure 3. Simple Traversal Algorithms

Shader Toy Демо

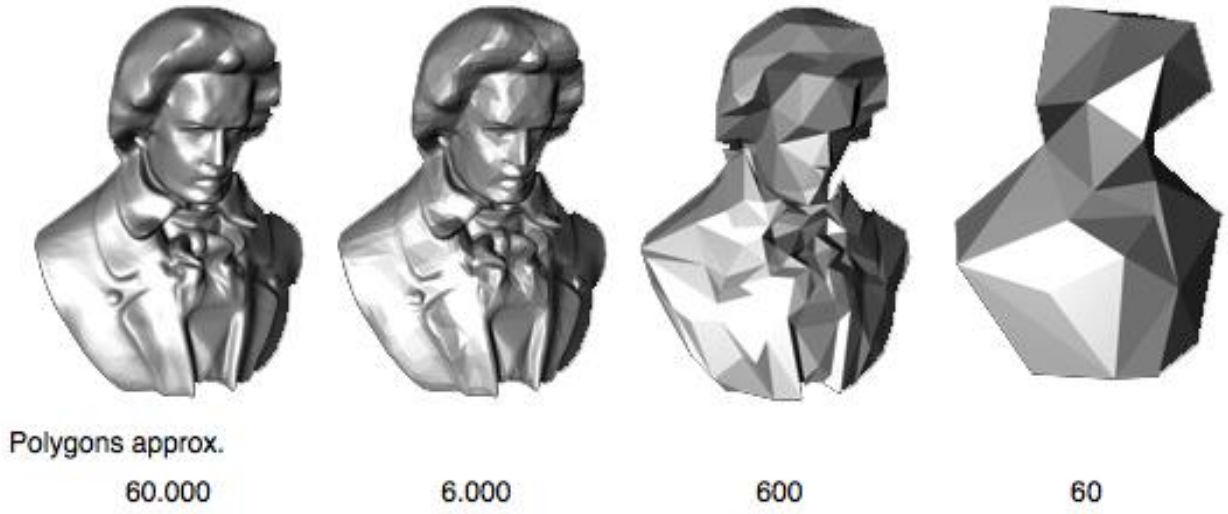
Много триъгълници

- 2 буфера с информация
  - Върtекс Буфер - Информация за точките?
  - Индекс Буфер – Кои точки правят триъгълници?

Shader Toy Демо 2

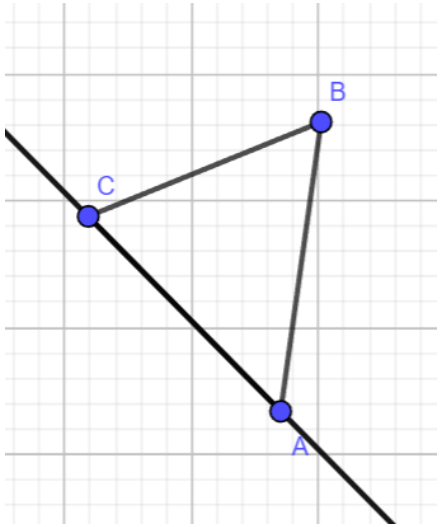
- **От къде идва тази информация?**
  - Идва от Артистите, които правят 3Д модели за нас.
  - Трябва да се зареди от някакъв формат
    - OBJ – Прост и лесен
    - glTF 2.0 – Прилича на JSon, (сравнително) лесен за четене от хора
    - USD – Pixar

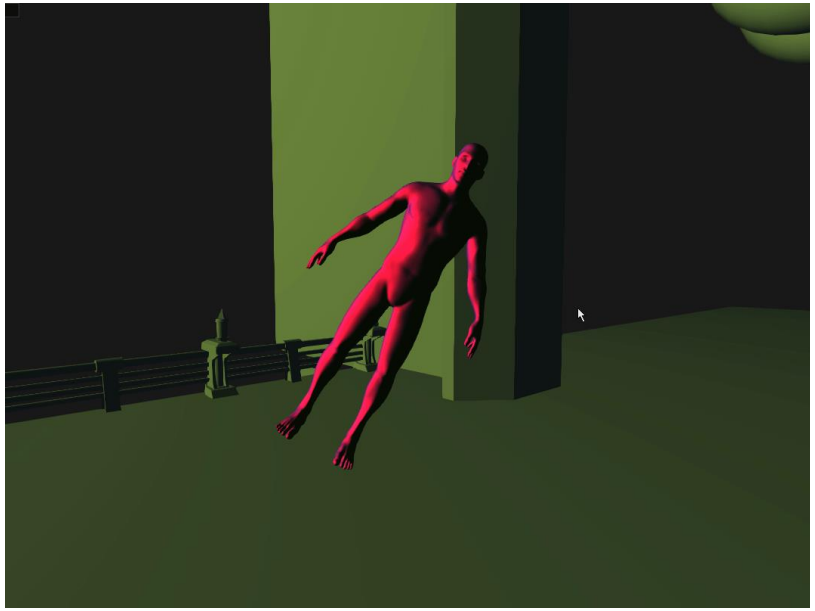
- Защо триъгълници?



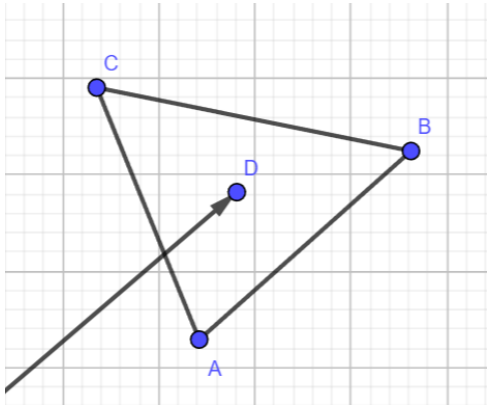
Как използваме Vertex-ите:

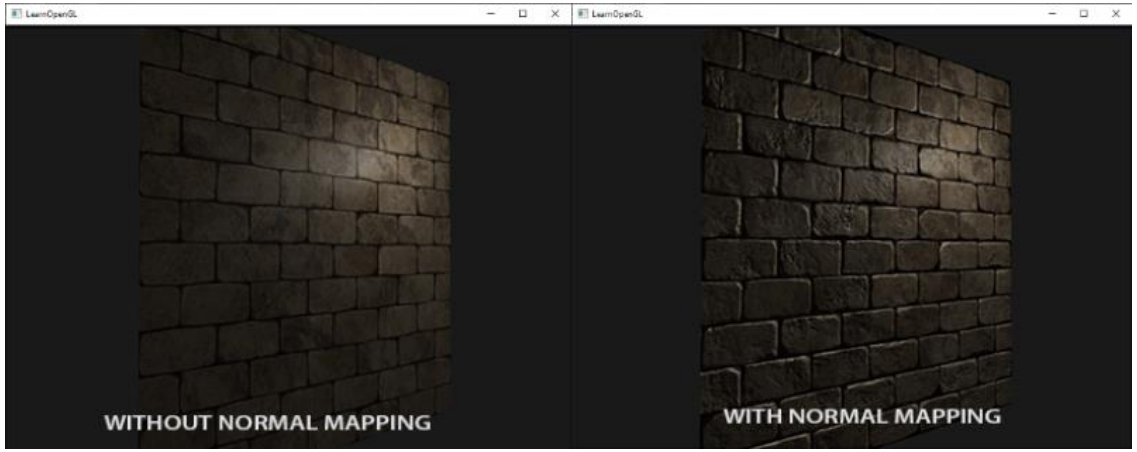
- В един върtекс може да имаме повече информация:
  - Позицията на върtекса
  - Текстурни Координати
  - Нормала
  - Тангентата на нормала
  - Битангентата на нормала
  - Цвят
  - ...и др.





- Как да разберем кой върtекс колко влияние трябва да има към някакъв пикксел?

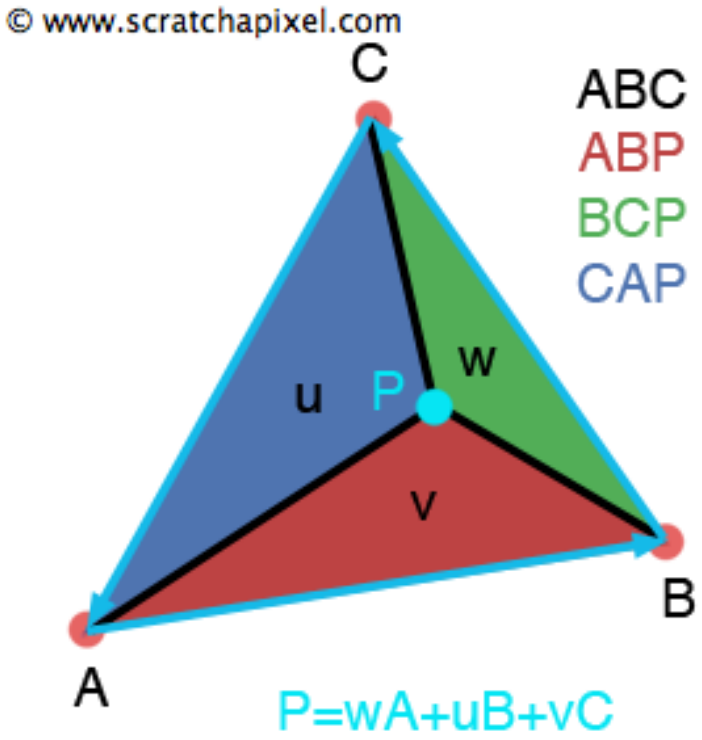




Barycentric Координати

- Функция, която ни връща 3 числа: X, Y и Z
  - Представлява, колко влияние има всеки въртекс, който имаме.
  - X + Y + Z = 1
  - Изчислява се от повърхнината на всеки от триъгълниците, които се получава, заедно с пиксела, който изчисляваме.

$$1 = \lambda_1 + \lambda_2 + \lambda_3$$
$$\vec{p} = \lambda_1\vec{v}_0 + \lambda_2\vec{v}_1 + \lambda_3\vec{v}_2$$




```
X = (повърхнина на триъгълник PBC) / (повърхнина на триъгълник ABC)
Y = (повърхнина на триъгълник APC) / (повърхнина на триъгълник ABC)
Z = (повърхнина на триъгълник ABP) / (повърхнина на триъгълник ABC)
```

```
64 vec3 barycentrics(vec2 p, vec2 a, vec2 b, vec2 c){
65     vec2 v0 = b - a, v1 = c - a, v2 = p - a;
66     float den = v0.x * v1.y - v1.x * v0.y;
67     float x = (v2.x * v1.y - v1.x * v2.y) / den;
68     float z = (v0.x * v2.y - v2.x * v0.y) / den;
69     //Няма смисъл да го изчисляваме трето ако
70     // x + y + z = 1.0
71     float y = 1.0f - x - z;
72
73     return vec3(x, y, z);
74 }
```

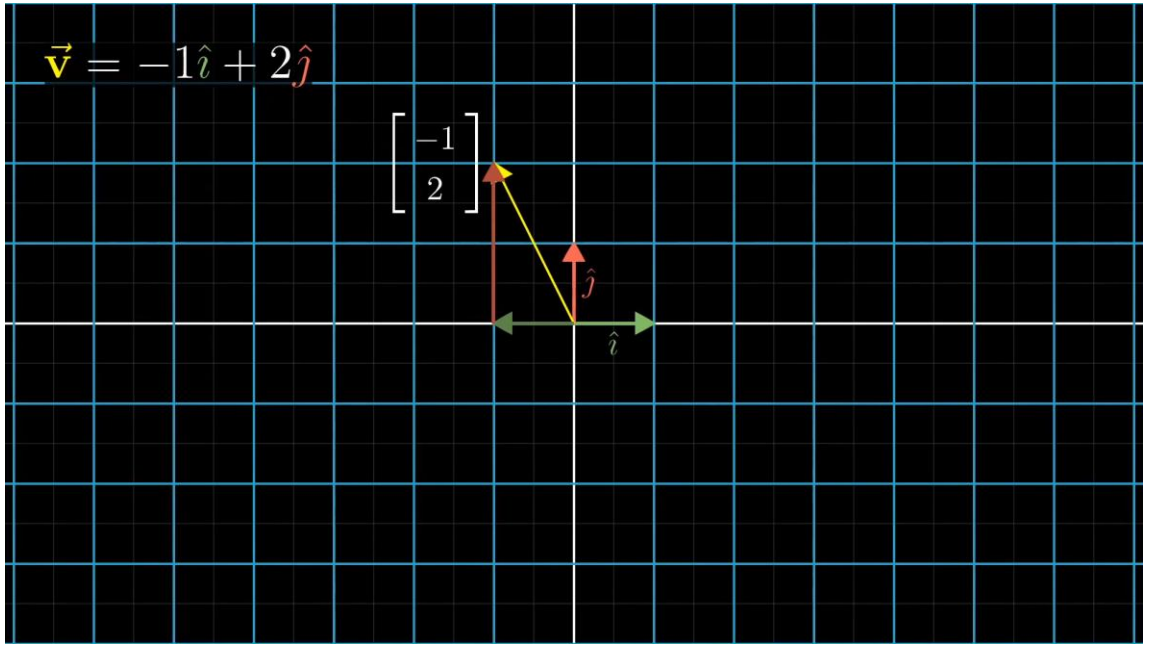
- GeoGebra пример – автор: Lin Lou

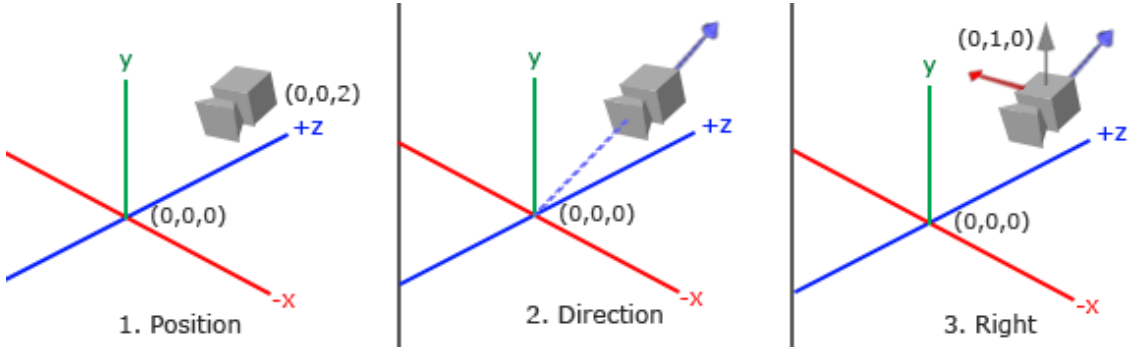
Shader Toy Демо 3

Камера:

- Не съществува.
- Само информация за нейните свойства
- Каква ни е целта с камерата?
  - Да изместим света, така че да е „пред нас“
  - Да го сложим в перспектива (близко = голямо, далече = малко)
- За да постигнем това използваме 4x4 матрици. (за 3Д)
  - И 3x3 матрици за 2Д.

Shader Toy Демо 4





X-Rotation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-Rotation in 3D

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale in 3D

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(4x4)*(4x1) = (4x1)

Y-Rotation in 3D

$$\begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix Multiplication

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ q \end{bmatrix}$$

View Матрицата:

```cpp
glm::mat4 CameraMatrix = glm::lookAt(
    cameraPosition, // the position of your camera, in world space
    cameraTarget,   // where you want to look at, in world space
    upVector        // probably glm::vec3(0,1,0), but (0,-1,0) would make you looking upside-down, which can be great too
);
```

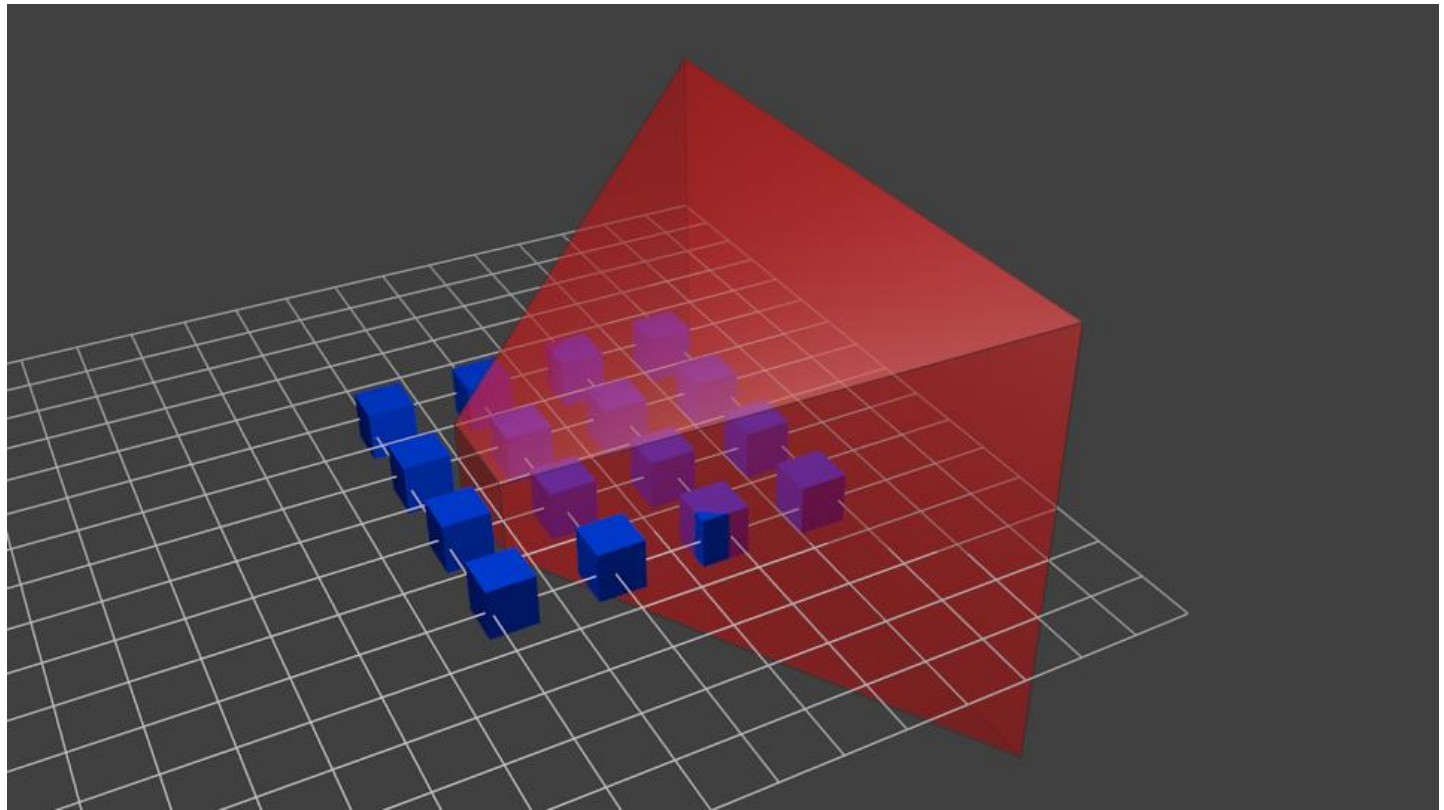
Всички библиотеки за математика имат функция.

Най – известната е GLM.

OpenGL Mathematics.

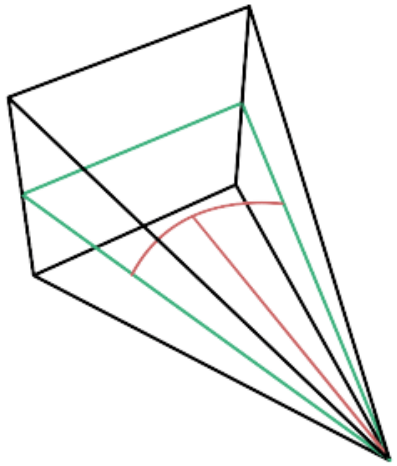
- Не е нужно да разбираме КАК го прави, само какво прави
  - И как се работи с нея

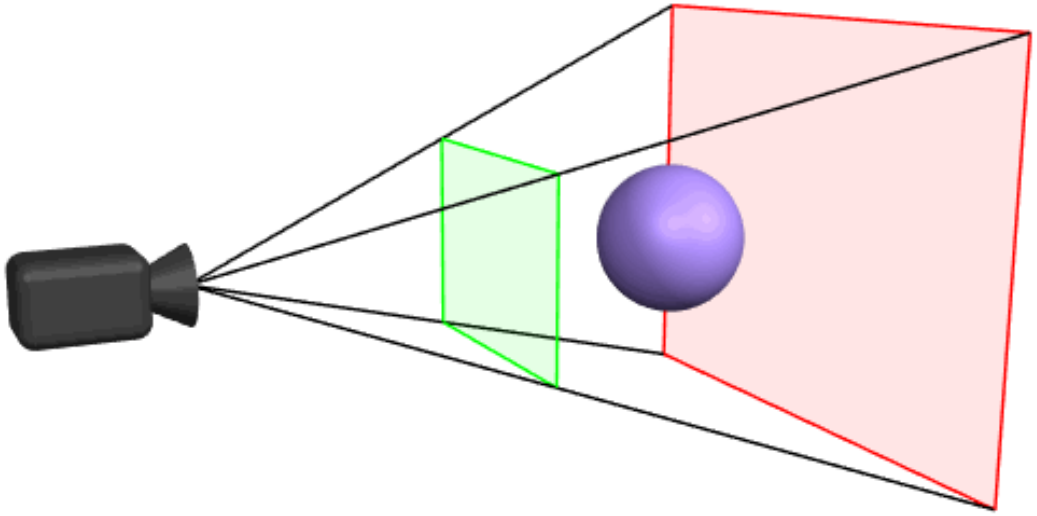
- Мести света около камерата и го слага така че всички обекти да са пред нея.

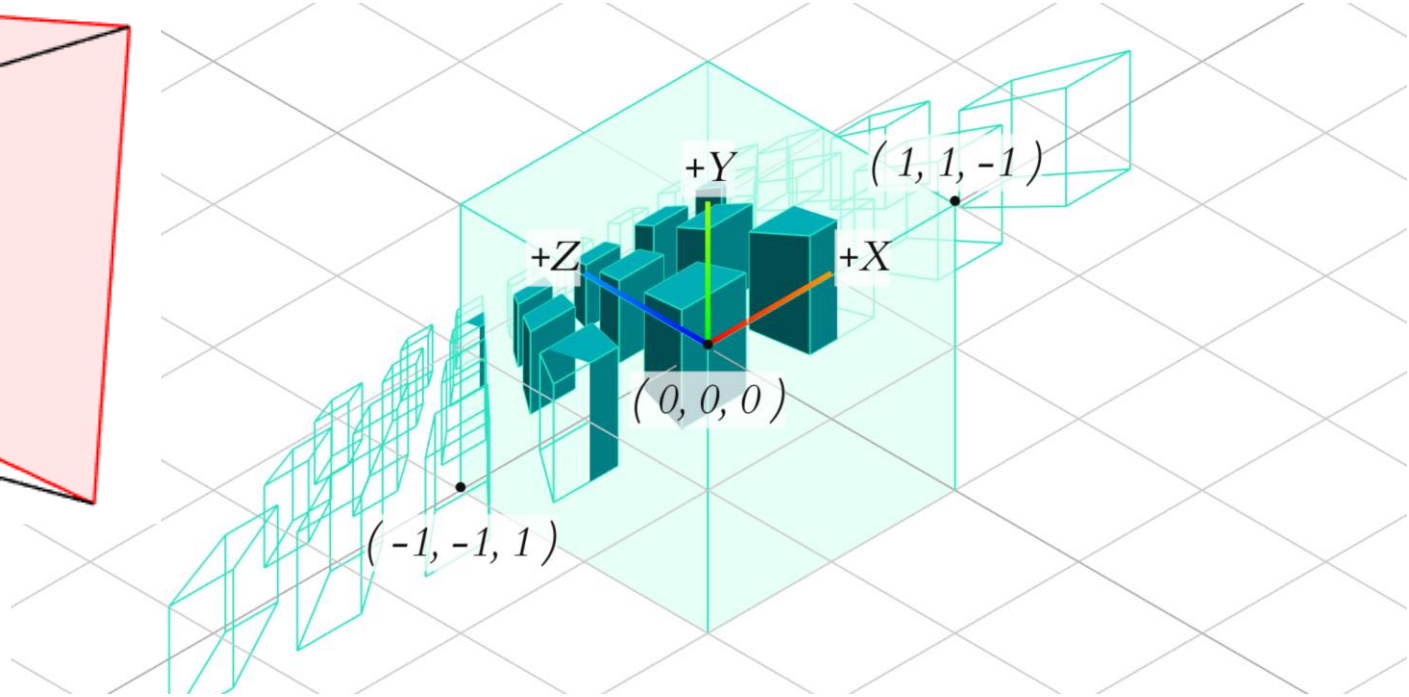


Projection Матрицата:

```cpp
// Generates a really hard-to-read matrix, but a normal, standard 4x4 matrix nonetheless
glm::mat4 projectionMatrix = glm::perspective(
    fieldOfView,  // Field of View
    aspectRatio,  // Depends on the size of the Window - 1980/1020
    nearClippingPlane, // Near clipping plane
    farClippingPlane   // Far clipping plane
);
```



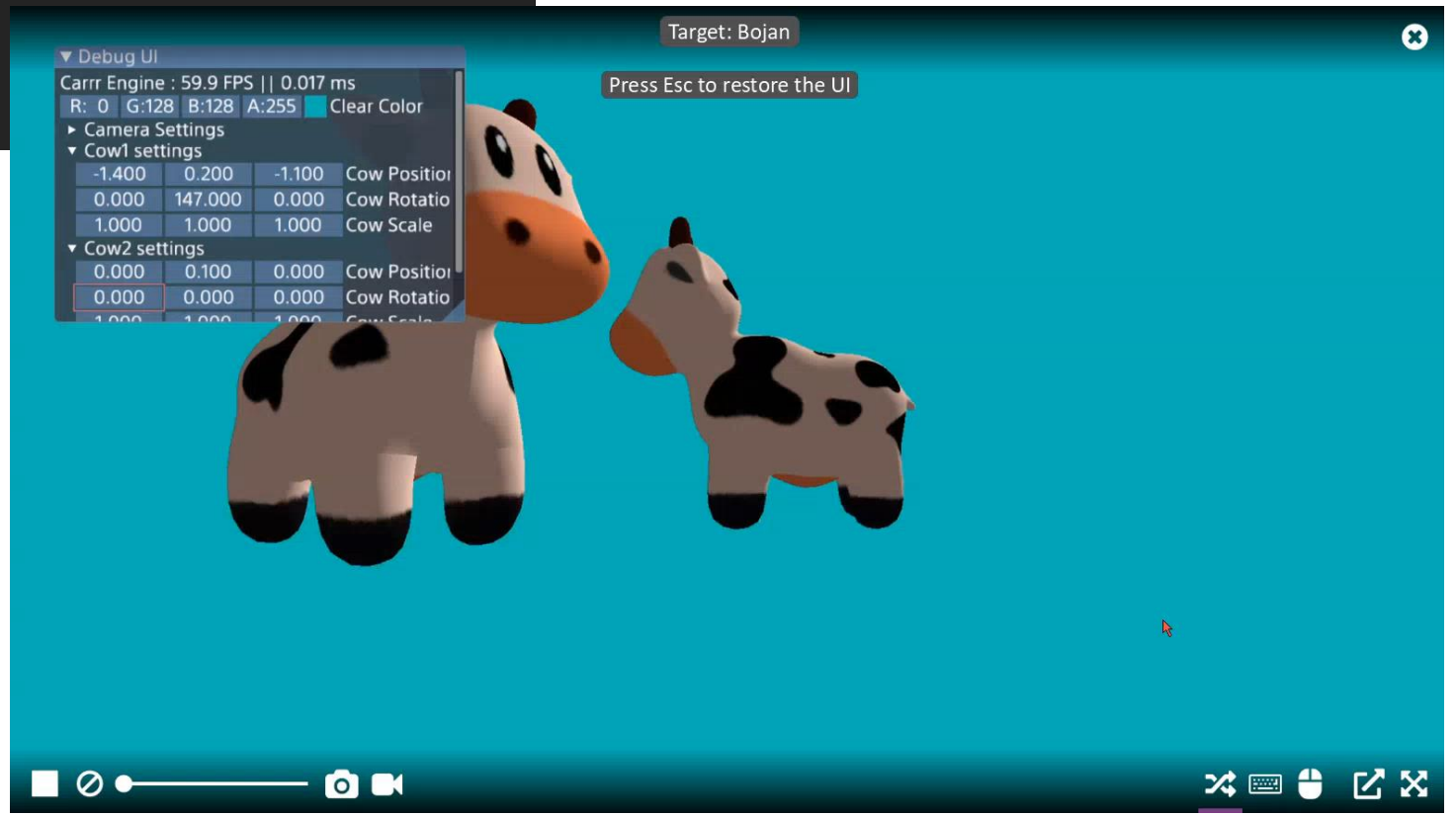




Модел Матрицата:

```cpp
// Мести модел
glm::mat4 translate = glm::translate(mat4(1.0f), vec3(0.5f, 0.5f, 0.0f));

// Върти модел
glm::mat4 rotate = glm::rotate(mat4(1.0f), 90.0f, vec3(0.0f, 0.0f, 1.0f));

// Уголемява модел
glm::mat4 scale = glm::scale(mat4(1.0f), vec3(0.5f, 0.5f, 1.0f));

// Комбинираме всичко заедно в 1 матрица
glm::mat4 m = translation * rotation * scale;
// Трябва да са в този ред translation * rotation * scale !!
```

```cpp
// Our ModelViewProjection : multiplication of our 3 matrices
glm::mat4 mvp = Projection * View * Model; // Remember, matrix

// Remember, matrix multiplication is the other way around
```

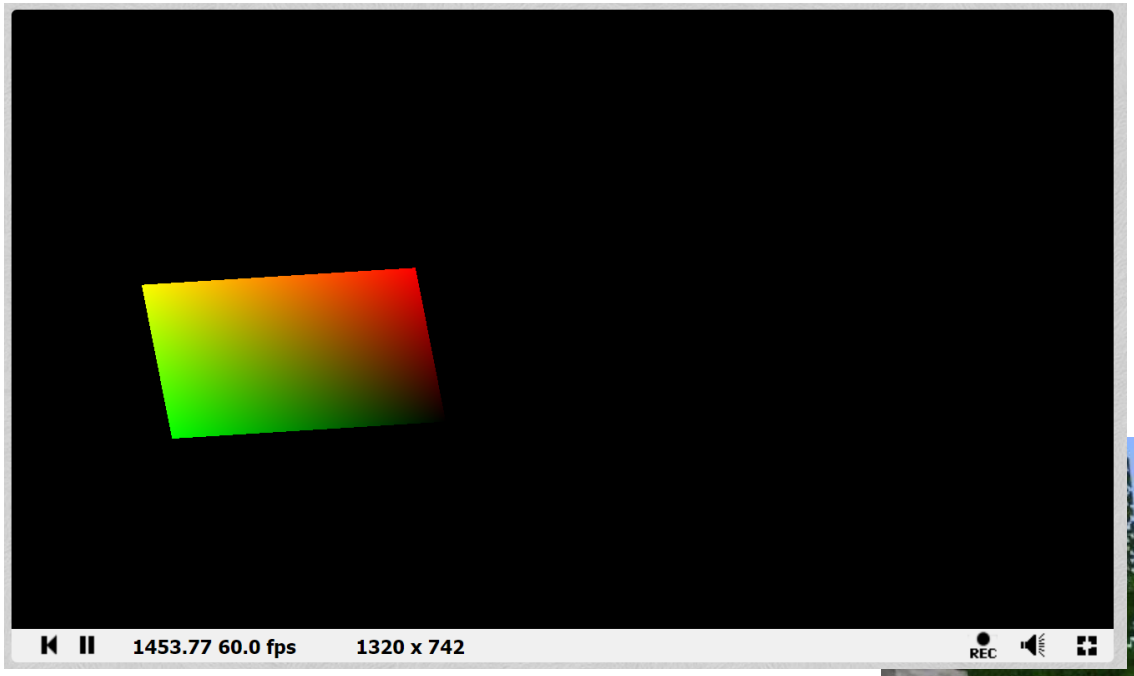


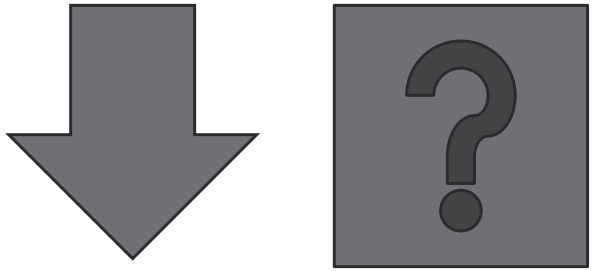
- 1 матрица за всеки модел (група от триъгълници)

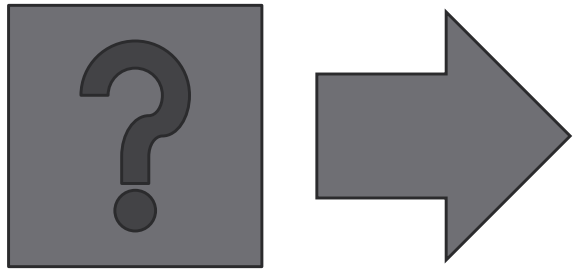
- Дава ни контрол на:
  - Позицията
  - Завъртането
  - Размера

Shader Toy Демо 5

Какво ни липсва? :







Текстури :

- Снимка, която живее в GPU-то
  - Там където Vertex и Index буферите

- Финалното ни изображение, също е текстура

Как се използва?

- **Текстурните координати**
  - Най – често от 0 до 1
  - определят от коя част на снимката / пиксел, ще вземем информацията.

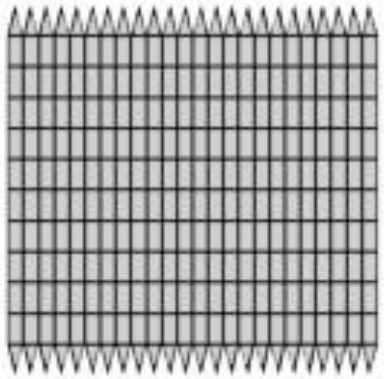
Защо текстури, а не буфер от инфо?

- По – бърз достъп до инфо
- По – бързо сампълване
- По – бързо филтриране

Текстури :

3-D Model

UV Map



[Shader Toy Демо 5 (отново)]

$p = (x,y,z)$

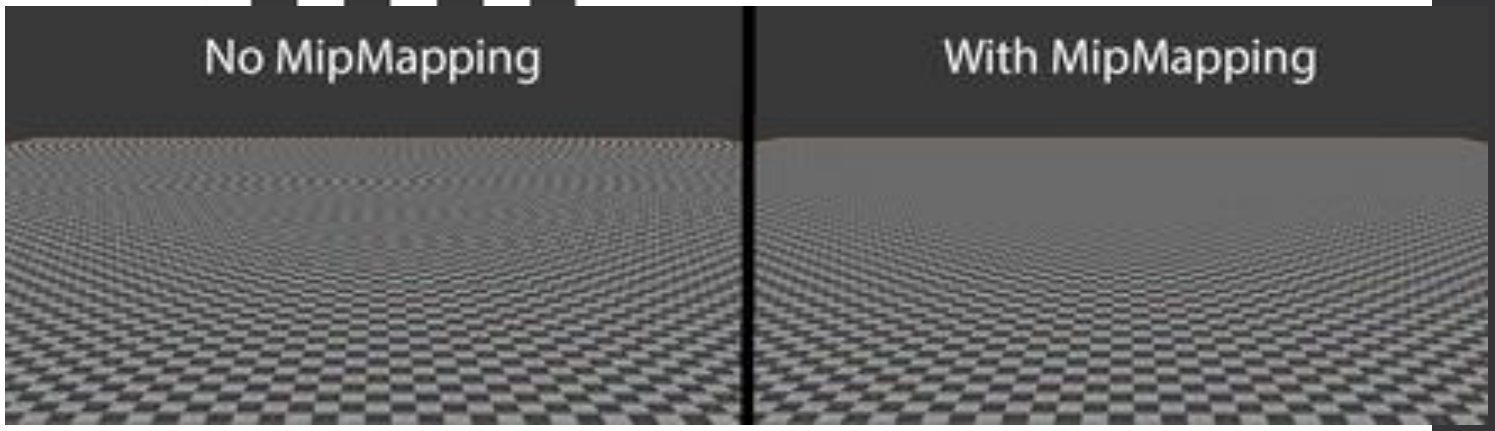
$p = (u,v)$

Texture

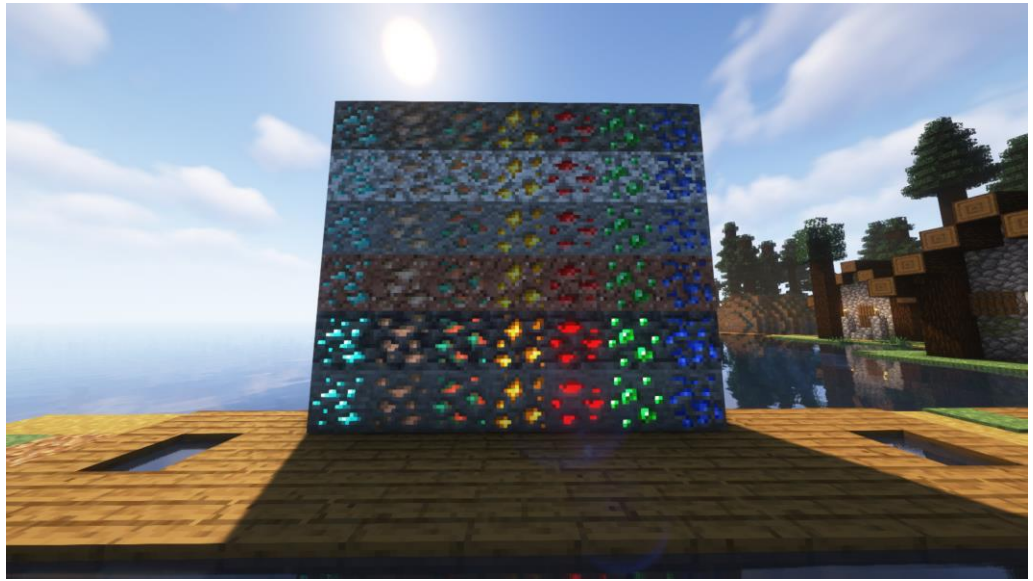
Още за Текстури :

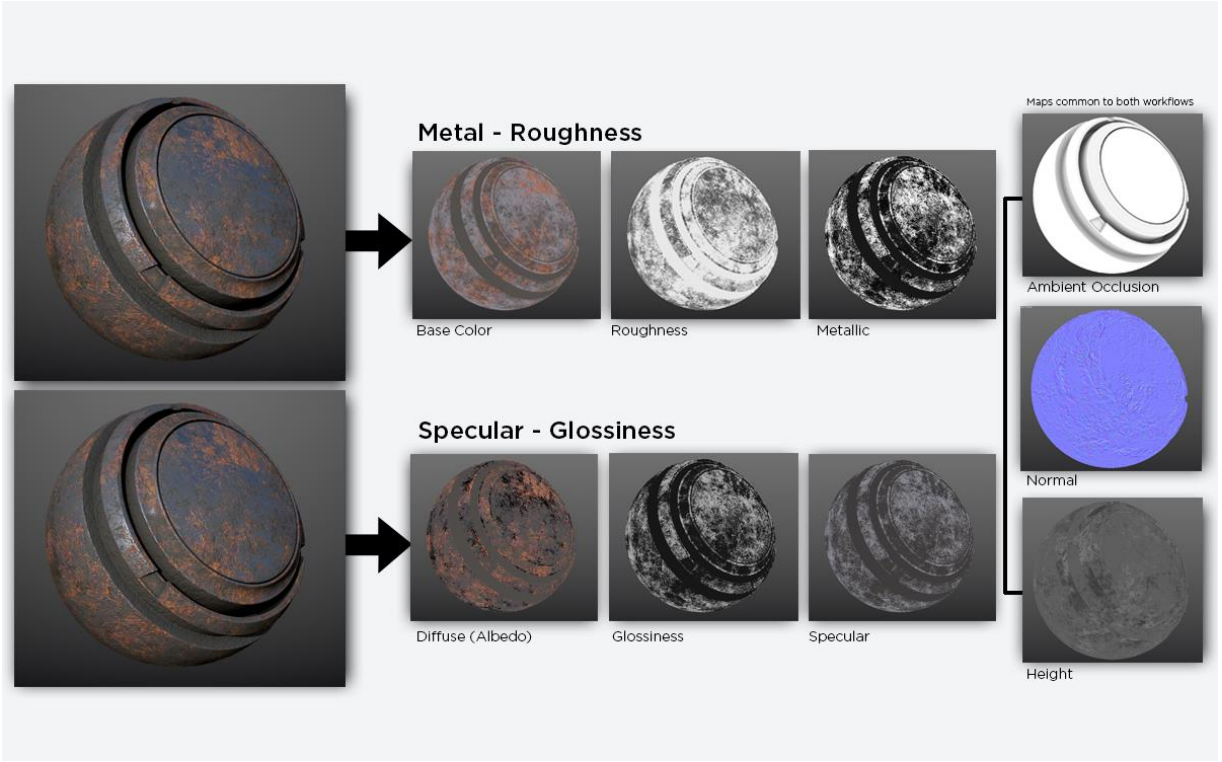
Shader Toy Демо 5 (пак пак)

- **Мип – вериги**
  - Същата текстура запазена няколко пъти
  - По – бърз сампълинг и по – красиви картини



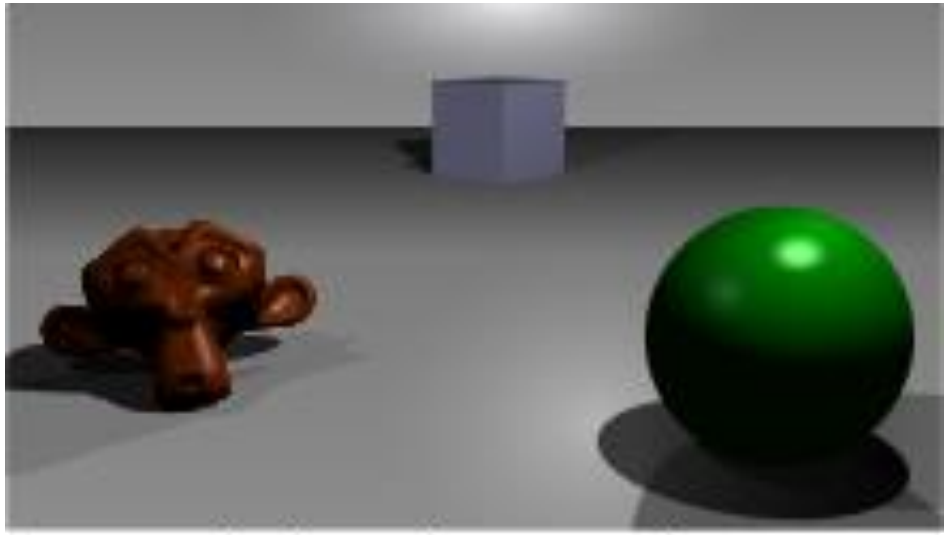
- **Не са само за цвят!**
  - В тях се запазва каквато и да е информация.





Дълбочина :

- **Z - буфер**
  - Служи за проверка „Кой триъгълник е най - отпред“
  - Извършва се след MVP трансформацията.



A simple three-dimensional scene

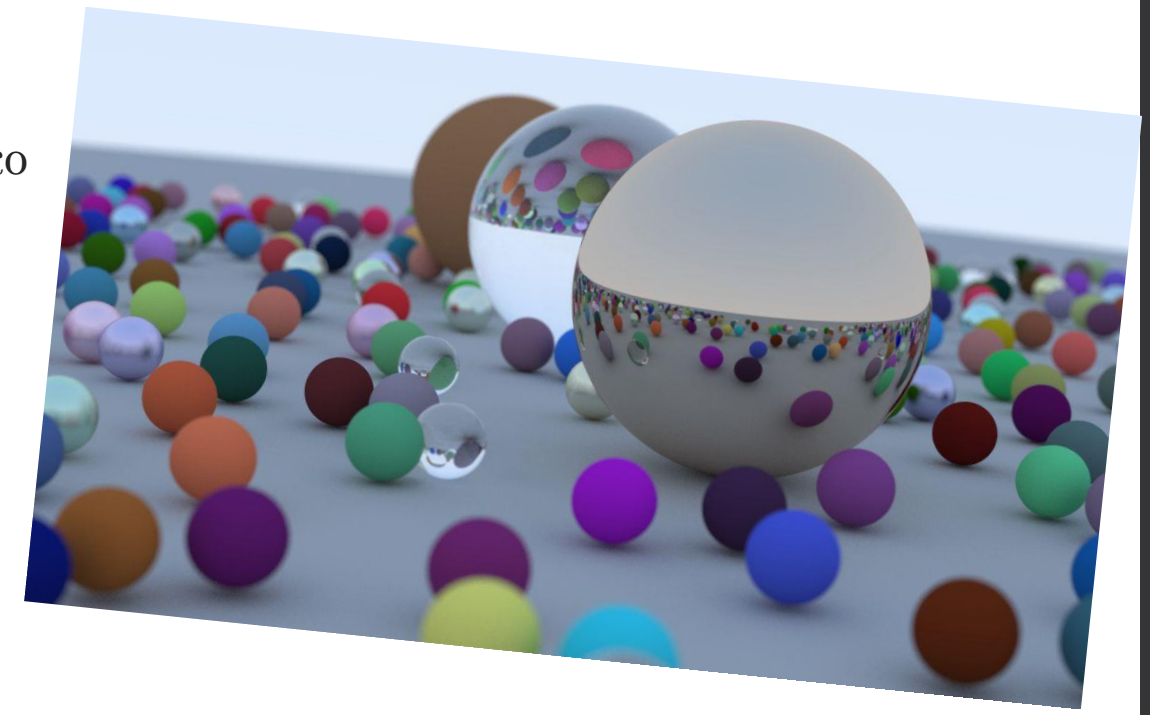
Z-buffer representation

Doom in Shader Toy

Още ресурси :

- LearnOpenGL –
  - От нищо до 3Д Рендерър на GPU-то.
  - Графично API за начинаещи

- Ray Tracing in One Weekend
  - От нищо до CPU Ray Tracer
  - Забавно предизвикателство за уикенда++
  - Има още 2 книги, които навлизат по - дълбоко



- Scratch Pixel Website –
  - 3D Rendering – Rasterization and Ray Tracing
  - Математика
  - “Процедурно“ Създаване на 3Д Светове

Снимка:
“Finding Your Home in Game Graphics Programming”
- Alex Tardif

Благодаря за вниманието!

Време за въпроси?